COS 425:
Database and Information
Management Systems

Query Evaluation:
Beyond Joining

1

Selection

- Operating on only one relation (file)
- Worst case: sequential search
  - Linear time
  - Often best case too
- If have index on R.f?
  - Equality condition on R.f
    => look up cost of index
  - Range lb ≤ R.f ≤ ub condition and tree index
    => look up cost of index

2

Selection with multiple conditions

R.x = a AND (R.y = b OR R.z < c) …

- Linear search: check Boolean expression of all conditions at once
  - No extra cost – all in main memory
- If have indexes on fields in selection
  - AND of conditions:
    - use index giving lowest cost to retrieve candidates satisfying condition on field of index
      - Cost to retrieve record?
      - Number of records retrieve?
    - Check other conditions on retrieved records

3

Selection with multiple conditions

continued

- If have indexes on fields in selection
  - OR of conditions:
    1. Retrieve records satisfying *each and every* condition using index
    2. Union retrieved sets to form result of OR
    - ❖ Total cost of 1. must be less than *one* linear scan
    - ❖ If any field used in condition has no index must do scan

      => *only* do scan

4

Selection with multiple conditions AND indexes giving *record pointers**

If index for *every* field involved => alternative algorithm:

1. *For each equality or inequality condition*
   Retrieve using index, the pointers (record IDs) for records satisfying condition
2. Sort sets of pointers
3. Merge sets of pointers
   - For AND, take *intersection*
   - For OR, take *union*
4. Retrieve actual data records using pointers

Must evaluate if will be cheaper than getting data records earlier in process

* i.e. “Alternative 2” [R&G] indexes or secondary-type indexes

5

Using record pointers

- If can get pointers for all records in query result can look up data records once
- Manipulate pointers of candidate records
  - Smaller size
- When ready to retrieve data records
  - Sort disk page location of pointers
    - Result may be much smaller than relation
  - Read each disk page once
  - Read disk pages contiguously

6

Projection

- Must read all records – linear scan
- Only issue is duplicate removal
  1. Most common technique: Sort
     - Can eliminate unwanted fields in Stage 1 of sort
       - Shrinks record size => less pages to write (maybe)
     - Can eliminate duplicates in merge phases of sort
  2. Alternate technique: analogous to hash-join
     1. Drop fields don't want and hash into F-1 buckets
     2. For each bucket
        1. If bucket fits in F-1 buffer pages, eliminate duplicates
        2. Otherwise, recurse
  3. Gift: sorted file on multi-field sort key and fields want are a prefix
     - When eliminate unwanted fields, duplicates adjacent

7

COS 425:
Database and Information
Management Systems

Query Optimization

8

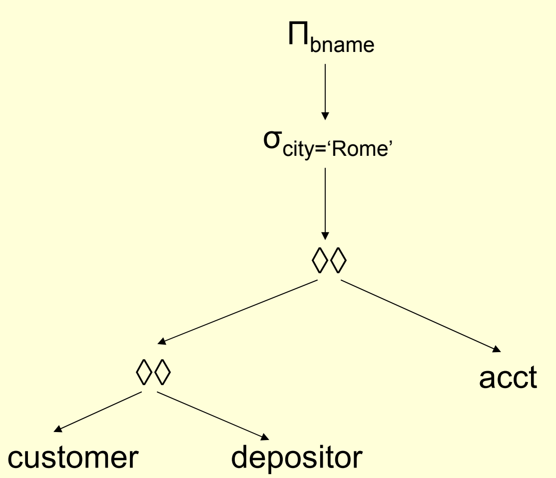
Query Optimization

- Query as expression over relational algebraic operations
- Get evaluation (parse) tree
  - Leaves: base relations
  - Interior nodes: operations

9

Example

$\Pi_{bname} (\sigma_{city='Rome'} ((customer \lozenge\lozenge depositor) \lozenge\lozenge acct))$

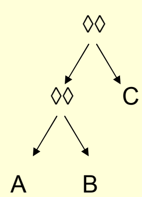


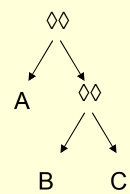
10

Optimization considerations

- Choice of algorithm at each interior node
  - Cost Estimates
    - We've just studied analysis
- Rearrange tree
  - Use algebra of operations
    - e.g. associativeness of JOIN

(A ◊◊ B) ◊◊ C
=
A ◊◊ (B ◊◊ C)



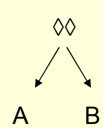


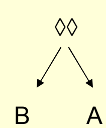
11

Interaction of algorithm choice and tree arrangement

- Convention: for any nested loop join, left branch represents outer relation
  - Control with commutivity of JOIN

    (A ◊◊ B) = (B ◊◊ A)





- Result of an interior node is input to parent
  - Algorithm affects properties of presentation of result
    - -Sorted?
- Cost analysis must proceed bottom up

12

Issues

- Need size estimates of result relation
  - # records per page (size of record)
  - # of pages (# of records)
  - Note:
    - page size fixed system parameter
    - Duplicates significantly affect # of records
- Need plan for buffer use
  - Write out all results of interior nodes to disk
    - Costs of writes for intermediate results count!
  - Intermediate result fits in buffer
    - Algorithm for parent use this?
    - Can save cost of writing result by child AND reading result by parent
  - Pipeline result of child as input to parent

13

Pipelining

- Parent and child execute concurrently
- Parent and child share buffer space
  - k-page shared (sub)buffer
  - child produces k pages of output – Fill buffer
  - parent consumes k pages of input from child – Empty buffer
  - NO disk write cost child;
  - NO disk read cost parent
- Algorithms of child and parent must support this
  - Child: usually does; produce 1 page output at a time
  - Parent: choice of algorithm critical !

14

Algorithms for parent - JOIN

- Block nested loop?
- Index nested loop?
- Sort-merge
- Hash

15

Algorithms for parent - JOIN

- Block nested loop?
  - Outer relation – ok
    - Read relation once buffer block by buffer block
    - Shared buffer becomes block
  - Inner relation – NO
    - Must re-read *entire* inner relation for every block outer
- Index nested loop?
- Sort-merge
- Hash

16

Algorithms for parent - JOIN

- Block nested loop?
  - Outer relation – ok
    - Read relation once buffer block by buffer block
    - Shared buffer becomes block
  - Inner relation – NO
    - Must re-read *entire* inner relation for every block outer
- Index nested loop?
  - Outer relation – ok – same as Block nested loop
  - Inner relation – NO
    - Using index
- Sort-merge
- Hash

17

Algorithms for parent - JOIN

- Block nested loop?
  - Outer relation – ok
  - Inner relation – NO
- Index nested loop?
  - Outer relation – ok – same as Block nested loop
  - Inner relation – NO
- Sort-merge
  - To sort input relation:
    - Can pipeline from child to block of buffers for Stage 1 (Stage 1: sorting individual blocks)
  - If child produces in sorted order, pipeline merge
    - Child must be outer relation if duplicates
- Hash

18

Algorithms for parent - JOIN

- Block nested loop?
  - Outer relation – ok
  - Inner relation – NO
- Index nested loop?
  - Outer relation – ok – same as Block nested loop
  - Inner relation – NO
- Sort-merge
  - To sort input relation:
    - Can pipeline from child to block of buffers for Stage 1 (Stage 1: sorting individual blocks)
  - If child produces in sorted order, pipeline merge
    - Child must be outer relation if duplicates
- Hash
  - To partition input relation:
    - Can pipeline from child to block of buffers for Stage 1

19

Allocating buffers

- If have simultaneous pipelining up tree
  - How many buffers for each child to parent exchange?
  - Affects speed of algorithms
- Limit number of simultaneous pipelines
- If no pipeline between child and parent materialize result of child
  - Child writes result to disk
  - Parent reads from disk

20

Multi-operation query

- Want plan
  - parse tree
  - Pipelining plan for each edge
  - Algorithm for each interior node (operation)
- To build plan
  - Consider alternatives
    - ALL?
  - Estimate costs
  - Choose "best"
    - Really "good enough"

21

Catalog

- Need info about base relations
- In catalog:
  - For each base relation:
    - # tuples
    - # pages
  - List of existing indexes
  - For each index
    - # distinct search-key values
    - # pages
  - For each tree index
    - Tree height
    - high/low search keys

22

Calculating size estimates of result

- Assume
  - independence of fields of a tuple
  - Uniform distribution of values of each field among tuples
- Calculate reduction factor (RF) for # tuples of result
  - **Examples:**

    $\sigma_{field = constant}$ and index on field:

    RF = 1/(# search key values)

    $\sigma_{field > constant}$ and tree index on field:

    $$RF = \frac{\text{(high key value)} - \text{constant}}{\text{(high key value)} - \text{(low key value)}}$$

  - **Estimate # pages output as RF * (# pages input relation)**

23

Reduction factor of joins

- Estimate # tuples of (R◊◊ $_{R.f = S.f}$ S) as

  RF * (# tuples R) * (# tuples S)

  - Looking at join as selection on RXS

- Example: ◊◊ $_{R.f = S.f}$
  - If indexes on R.f and S.f

    RF = 1/max (# key values R.f, # key values S.f)
  - If no indexes, *could* use # distinct values
    - What if real-valued?

24

Size of tuples of result

- If fields of fixed length, calculate
  - Projection: sizes of fields retained
  - Cross-product RXS: sum of sizes of tuples in R and S
    - Join with single occurrence equal fields
      - Projection of Cross-product
  - Selection & Union-compatible set operations: no change
- If fields of variable length, estimate

25

Planning

- Know how estimate costs of algorithms
- Know how estimate sizes of results ON
- How use to make plan for query eval?

SKETCH ON BOARD

26